МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Построение и анализ алгоритмов»

Тема: Визуализация работы алгоритма Кнута-Моррисона-Прата

Студент гр. 3381 _____ Сычев Н.С.

Преподаватель _____ Шестопалов Р.П.

Санкт-Петербург

2025

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Сычев Н.С.

Группа 3381

Тема работы**:** Визуализация работы алгоритма Кнута-Моррисона-Прата

Исходные данные:
Тема: Визуализация алгоритма Кнута-Морриса-Пратта (КМП) поиска подстроки в строке.
Язык реализации: Go (основная логика), JavaScript (визуализация), HTML/CSS.

Содержание пояснительной записки: «Введение», «Теоретические основы задачи поиска подстроки», «Алгоритм Кнута-Морисса-Пратта», «Реализация инструмента визуализации КМП», «Заключение», «Приложение А. Исходный код программы»

Предполагаемый объем пояснительной записки:
Не менее 15 страниц.

Дата выдачи задания: 17.04.2025

Дата сдачи реферата: 22.04.2025

Дата защиты реферата: 22.04.2025

Студент _____ Сычев Н.С.

Преподаватель _____ Шестопалов Р.П.

АННОТАЦИЯ

Курсовая работа посвящена разработке системы визуализации алгоритма Кнута-Морриса-Пратта (КМП) для поиска подстроки на языках Go и JavaScript. Основной целью проекта является создание интерактивного инструмента, демонстрирующего пошаговое выполнение алгоритма, включая сравнение символов, сдвиги подстроки и динамическое построение префикс-функции. Для реализации функциональности были использованы структуры данных: массив для хранения префикс-функции, массив шагов для анимации и HTML Canvas для графической визуализации.

Система поддерживает следующие операции:

- Ввод текста и подстроки для поиска.
- Пошаговая визуализация процесса КМП с подсветкой текущих символов и префикс-функции.
- Регулировка скорости анимации и пошаговое выполнение.
- Вывод позиций совпадений и количества сравнений.
- Сохранение истории последних поисков в локальном хранилище браузера.

В процессе разработки применены принципы клиент-серверной архитектуры, алгоритмической оптимизации и веб-программирования для обеспечения высокой производительности и удобства использования. Реализация визуализации с использованием Canvas. Результатом работы является система, обеспечивающая наглядное представление работы алгоритма КМП.

SUMMARY

The course work is devoted to the development of a visualization system for the Knuth-Morris-Pratt algorithm (KMP) for substring search in Go and JavaScript languages. The main goal of the project is to create an interactive tool that demonstrates step-by-step algorithm execution, including character comparison, substring shifts, and dynamic prefix function construction. To implement the functionality, data structures were used: an array for storing the prefix function, an array of steps for animation, and an HTML Canvas for graphical visualization.

The system supports the following operations:

• Text input and substrings for search.

• Step-by-step visualization of the KMP process with highlighting of current symbols and prefix function.

• Animation speed adjustment and step-by-step execution.

• Output of match positions and the number of comparisons.

• Save the history of recent searches in the browser's local storage.

During the development process, the principles of client-server architecture, algorithmic optimization and web programming were applied to ensure high performance and ease of use. Implementation of visualization using Canvas. The result of the work is a system that provides a visual representation of the operation of the KMP algorithm.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Алгоритм Кнута-Морриса-Пратта (КМП) является одним из наиболее эффективных методов поиска подстроки в тексте, отличающимся линейной сложностью O(n+m), где n — длина текста, а m — длина подстроки. Его ключевая особенность — использование префикс-функции для минимизации повторных сравнений.

Данная курсовая работа посвящена разработке системы визуализации алгоритма КМП, реализованной на языках Go и JavaScript. Цель проекта — создание интерактивного инструмента, который позволяет пользователю вводить текст и подстроку, наблюдать процесс поиска подстроки в строке с анимацией сравнений и сдвигов, а также изучать динамическое построение префикс-функции. Система поддерживает пошаговый режим, регулировку скорости анимации и сохранение истории поисков, что делает ее удобной для обучения и анализа.

1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ЗАДАЧИ ПОИСКА ПОДСТРОКИ

Задача поиска подстроки является одной из ключевых в области компьютерных наук, особенно в обработке строк, анализе текстов и поиске шаблонов. Она заключается в нахождении всех вхождений заданной подстроки (шаблона) в более длинном тексте (строке). Эта задача имеет широкое применение в различных областях, таких как текстовые редакторы, поисковые системы, биоинформатика (например, сопоставление последовательностей ДНК), сжатие данных и анализ больших данных. Теоретические основы этой задачи включают формальное определение, анализ вычислительной сложности и разработку алгоритмов для ее эффективного решения.

1.1. Постановка задачи

Функциональные требования описывают конкретные возможности системы, которые обеспечивают выполнение ее основной задачи — визуализации алгоритма КМП, а также предоставляют пользователю удобный и интерактивный интерфейс для изучения и анализа работы алгоритма. Система должна поддерживать следующие функции:

Формальная постановка

- Текст T, шаблон P и алфавит из которого будет состоять T и P

  Требуется:
- Найти все индексы i (где $0 \le i \le n-m$), такие что подстрока $T[i:i+m]=P$
- Если вхождений нет, вернуть пустой список или сообщение об отсутствии совпадений.

В реальных приложениях задача поиска подстроки часто усложняется дополнительными требованиями:

- Нечеткий поиск: Допускаются небольшие ошибки в шаблоне (например, опечатки).
- Многошаблонный поиск: Поиск нескольких шаблонов одновременно (решается, например, алгоритмом Ахо-Корасик).

- Регулярные выражения: Поиск шаблонов, заданных регулярными выражениями, что требует более сложных алгоритмов.

1.2. Наивный алгоритм поиска подстроки

Наивный алгоритм поиска подстроки — это простейший подход к решению задачи, который не использует предварительной обработки данных и полагается на прямое сравнение символов. Несмотря на свою простоту, он демонстрирует высокую вычислительную сложность, что делает его неэффективным для больших текстов.

Наивный алгоритм работает следующим образом:

1. Для каждого возможного индекса i в тексте T (от 0 до n−m) выполняется проверка, совпадает ли подстрока T[i:i+m] с шаблоном P.
2. Для каждой позиции i сравниваются символы T[i+j] и P[j] для всех j от 0 до m−1.
3. Если все символы совпадают, индекс i добавляется в список вхождений.
4. Если на какой-то позиции j обнаруживается несовпадение, проверка для текущего i прекращается, и алгоритм переходит к следующему индексу i+1.

1.3. Необходимость эффективных алгоритмов

Наивный алгоритм поиска подстроки — это простейший подход к решению задачи, который не использует предварительной обработки данных и полагается на прямое сравнение символов. Несмотря на свою простоту, он демонстрирует высокую вычислительную сложность, что делает его неэффективным для больших текстов.

Проблемы наивного подхода:

1. Высокая временная сложность: Как было показано, наивный алгоритм имеет сложность O(n·m), что неприемлемо для больших текстов. Например, для текста длиной n=10^6 и шаблона длиной m=10^ потребуется до 10^9 операций.

2. Избыточные сравнения: Наивный алгоритм не использует информацию о структуре текста или шаблона, повторно сравнивая символы, которые уже были проверены.
3. Скалируемость: В реальных приложениях, таких как поиск в базах данных или анализ геномных данных, объемы данных могут быть огромными, и наивный алгоритм становится непрактичным.

Алгоритм Кнута-Морриса-Пратта, реализованный в предоставленном приложении , является примером эффективного решения.

2. АЛГОРИТМ КНУТА-МОРИССОНА-ПРАТТА

Алгоритм Кнута-Морриса-Пратта (KMP) — это эффективный алгоритм для поиска подстроки в тексте, разработанный в 1977 году Дональдом Кнутом, Воном Праттом и Джеймсом Моррисом. Он решает задачу поиска всех вхождений шаблона P длиной m в текст T длиной n с временной сложностью O(n+m), что значительно лучше, чем O(n*m) наивного алгоритма. KMP использует префикс-функцию, которая позволяет минимизировать количество сравнений, избегая повторных проверок уже сопоставленных символов.

Основные характеристики:

Предобработка: Алгоритм анализирует шаблон и строит префикс-функцию за O(m).

Поиск: Выполняется за O(n), не возвращаясь к уже проверенным символам текста.

Применение: Текстовые редакторы, поисковые системы, биоинформатика, сжатие данных.

2.1. Основная идея алгоритма

Основная идея KMP заключается в использовании информации о структуре шаблона для избежания ненужных сравнений. Наивный алгоритм при несовпадении сдвигает шаблон на одну позицию и начинает сравнение заново, что приводит к повторным проверкам. KMP решает эту проблему с помощью префикс-функции, которая указывает, как сдвигать шаблон при несовпадении, чтобы продолжить сравнение с учетом уже найденных совпадений.

Проблема наивного подхода:

Наивный алгоритм для каждой позиции i в тексте T сравнивает все символы шаблона P. При несовпадении он сдвигает шаблон на одну позицию, игнорируя информацию о предыдущих совпадениях. Это приводит к временной сложности O(n*m).

Пример:

- Текст: T = "AAAAAB"
- Шаблон: P = "AAAB"
- Наивный алгоритм заново сравнивает символы, которые уже были проверены, что неэффективно.

Решение KMP

KMP использует префикс-функцию pi, где pi[j] — длина наибольшего суффикса P[0:j+1], равного префиксу P. При несовпадении T[i] P[j] , алгоритм сдвигает шаблон, устанавливая (j = pi[j-1]), что позволяет пропустить уже проверенные совпадения.

Процесс работы

1. Построение префикс-функции: Вычисляется массив pi за O(m).
2. Поиск:
    - Указатели: (i) (текст), (j) (шаблон).
    - Если (T[i] = P[j]), увеличиваем (i) и (j).
    - Если (j = m), найдено вхождение на (i - j).
    - Если (T[i] != P[j]):
        - Если (j = 0), увеличиваем (i).
        - Если (j > 0), устанавливаем (j = pi[j-1]).

Пример

Текст: (T = "AABABC"), шаблон: (P = “ABC”).

- Префикс-функция: (pi = [0, 0, 0]).
- Поиск:
    - (i = 0, j = 0): (T[0] = “A“ = P[0]). (i = 1, j = 1).
    - (i = 1, j = 1): (T[1] = “A“ != P[1]). (j = pi[0] = 0).
    - (i = 5, j = 2): (T[5] = “C“ = P[2]). (j = 3), вхождение на (i - j = 2).
- Результат: Вхождение на позиции (2).

2.2. Префикс-функция

Префикс-функция — ключевой компонент KMP, который позволяет эффективно сдвигать шаблон при несовпадении. Для каждой позиции (j) в шаблоне P, pi[j] указывает длину наибольшего суффикса (P[0:j+1]), который является префиксом (P).

Построение

1. Инициализируем (pi[0] = 0).
2. Для (j) от 1 до (m-1):
    - (k = pi[j-1]).
    - Если (P[j] = P[k]), то (\pi[j] = k + 1).
    - Если (P[j] != P[k]), уменьшаем (k) до (pi[k-1]), пока (P[j] = P[k]) или (k = 0).
    - Если (k = 0) и (P[j] != P[0]), то (pi[j] = 0).

Пример

Шаблон: (P = "ABABAC").

- (j = 0): (pi[0] = 0).
- (j = 1): (P[1] = “B“ != P[0] = “A”), (pi[1] = 0).
- (j = 2): (P[2] = “A“ = P[0]), (pi[2] = 1).
- (j = 3): (P[3] = “B“ = P[1]), (pi[3] = 2).
- (j = 4): (P[4] = “A“ = P[2]), (pi[4] = 3).
- (j = 5): (P[5] = “C“ != P[3] = “B“), (pi[5] = 0).
- Итог: (pi = [0, 0, 1, 2, 3, 0]).

Использование

При несовпадении (T[i] != P[j]), (j = pi[j-1]), что сдвигает шаблон к следующей возможной позиции совпадения.

Преимущества

- Экономия сравнений за счет пропуска уже проверенных совпадений.
- Линейная сложность (O(m)) для построения.
- Универсальность для любого алфавита.

3. РЕАЛИЗАЦИЯ ИНСТРУМЕНТА ВИЗУАЛИЗАЦИИ КМП

Инструмент визуализации алгоритма Кнута-Морриса-Пратта (KMP) представляет собой веб-приложение, разработанное для демонстрации работы алгоритма поиска подстроки в тексте. Приложение позволяет пользователю вводить текст и шаблон, наблюдать пошаговое выполнение алгоритма, визуализировать сравнение символов и использование префикс-функции, а также изучать результаты поиска. Инструмент реализован с использованием клиент-серверной архитектуры, где клиентская часть (фронтенд) отвечает за пользовательский интерфейс и анимацию, а серверная часть (бэкенд) выполняет вычисления алгоритма KMP. Основная цель разработки — создание образовательного инструмента, который делает сложный алгоритм KMP доступным для понимания студентами и разработчиками.

Приложение решает следующие задачи:

- Обеспечение интерактивного интерфейса для ввода текста, шаблона и настройки скорости анимации.
- Визуализация каждого шага алгоритма, включая сравнение символов, сдвиги шаблона и значения префикс-функции.
- Предоставление пользователю возможности управлять анимацией (запуск, пауза, перемотка шагов вперед и назад).
- Отображение результатов поиска (позиций вхождений шаблона) и статистики (количество сравнений).

3.1. Архитектура приложения

Архитектура приложения основана на модели клиент-сервер, которая разделяет функциональность на две независимые части: клиентскую (фронтенд) и серверную (бэкенд). Такое разделение обеспечивает модульность, упрощает разработку, тестирование и потенциальное масштабирование приложения. Архитектура спроектирована так, чтобы пользователь мог взаимодействовать с приложением через веб-браузер, а сервер обрабатывал вычисления и предоставлял данные для визуализации.

Компоненты архитектуры

1. Клиентская часть (фронтенд):
    - Технологии: HTML для структуры страницы, CSS для стилизации интерфейса, JavaScript для обработки логики и Canvas API для отрисовки анимации.
    - Задачи:
        - Создание пользовательского интерфейса с полями ввода для текста и шаблона, кнопками управления и областью для визуализации.
        - Отрисовка текста, шаблона и префикс-функции на канвасе с подсветкой текущих позиций.
        - Обработка пользовательских действий: ввод данных, запуск/пауза анимации, перемотка шагов, настройка скорости.
        - Отправка запросов к серверу и обработка полученных данных для отображения шагов алгоритма.
    - Компоненты:
        - HTML-страница с элементами управления (поля ввода, кнопки, канвас).
        - CSS-стили для оформления, включая адаптивный дизайн и эффекты анимации.
        - JavaScript-код для взаимодействия с сервером, управления анимацией и обновления интерфейса.
2. Серверная часть (бэкенд):
    - Технологии: Язык программирования Go для реализации HTTP-сервера и логики KMP.
    - Задачи:
        - Обработка HTTP-запросов от клиента, включая получение текста и шаблона.

        - Выполнение алгоритма KMP с генерацией пошаговых данных (сравнения, сдвиги, значения префикс-функции).
        - Формирование ответа в формате JSON, содержащего шаги алгоритма, позиции вхождений и статистику.
    - Компоненты:
        - HTTP-сервер с двумя маршрутами: "/" \text{"/"} "/" для отдачи HTML-страницы и "/kmp" \text{"/kmp"} "/kmp" для обработки запросов на выполнение KMP.
        - Функции для построения префикс-функции и поиска подстроки с сохранением шагов.
        - Структуры данных для представления шагов и результатов.
3. Взаимодействие клиент-сервер:
    - Клиент отправляет POST-запрос на маршрут "/kmp" \text{"/kmp"} "/kmp" с JSON-объектом.
    - Сервер выполняет алгоритм KMP, генерирует шаги и возвращает JSON-ответ, содержащий:
    - Массив шагов с информацией о сравнениях, сдвигах и значениях префикс-функции.
    - Список позиций вхождений шаблона.
    - Общее количество сравнений.
    - Значения префикс-функции для шаблона.
    - Клиент обрабатывает ответ, отображая шаги на канвасе, обновляя префикс-функцию и показывая результаты в интерфейсе.

Поток данных

1. Пользователь открывает веб-страницу, которая загружается с сервера по маршруту "/" \text{"/"} "/".
2. Пользователь вводит текст и шаблон в соответствующие поля ввода и настраивает скорость анимации с помощью ползунка.
3. При нажатии кнопки "Start Animation" клиент отправляет POST-запрос на "/kmp" \text{"/kmp"} "/kmp" с входными данными.

4. Сервер выполняет алгоритм KMP, формирует массив шагов и возвращает JSON-ответ
5. Клиент использует полученные данные для анимации, отображая текст и шаблон на канвасе, подсвечивая текущие символы, обновляя префикс-функцию и показывая позиции вхождений.

Преимущества архитектуры

- **Модульность**: Разделение на фронтенд и бэкенд позволяет независимо разрабатывать и тестировать компоненты.
- **Интерактивность**: Пользователь может управлять анимацией, перематывать шаги и настраивать скорость, что улучшает образовательный процесс.
- **Масштабируемость**: Сервер можно расширить для поддержки других алгоритмов поиска подстроки (например, Бойера-Мура или Рабина-Карпа).
- **Кроссплатформенность**: Веб-приложение работает в любом современном браузере без необходимости установки дополнительного ПО.
- **Простота интеграции**: JSON-формат данных упрощает взаимодействие между клиентом и сервером.

Ограничения

- **Зависимость от сети**: Для работы требуется стабильное соединение с сервером, хотя в будущем можно реализовать выполнение KMP на стороне клиента.
- **Производительность**: Для очень длинных текстов обработка на сервере может замедлиться, что требует оптимизации.
- **Сложность визуализации**: Отображение длинных строк на канвасе может быть затруднено на устройствах с маленькими экранами.

3.2. Реализация бэкенд-части

Бэкенд реализован на языке Go и отвечает за выполнение алгоритма KMP, генерацию шагов и обработку запросов. Он включает структуры данных для хранения информации и функции для вычислений.

3.2.1. Структуры данных

Для представления состояния и результатов работы алгоритма KMP в бэкенде определены две основные структуры данных: Step и KMPResult. Эти структуры разработаны для хранения информации о каждом шаге выполнения алгоритма и общего результата, что позволяет фронтенду визуализировать процесс поиска подстроки.

Структура Step

Структура Step описывает один шаг выполнения алгоритма KMP. Она содержит всю необходимую информацию для отображения текущего состояния процесса поиска на фронтенде. Поля структуры включают:

- TextIndex (int): Индекс текущего символа в тексте, с которым проводится сравнение.
- PatternIndex (int): Индекс текущего символа в шаблоне (подстроке), с которым проводится сравнение.
- Match (bool): Флаг, указывающий, совпадают ли текущие символы текста и шаблона.
- Shift (bool): Флаг, указывающий, выполняется ли сдвиг шаблона в случае несовпадения.
- Status (string): Текстовое описание текущего шага (например, "Совпадение найдено" или "Несовпадение, сдвиг шаблона").
- FailureValue (int, опционально): Значение префикс-функции для текущего шага, используемое при сдвиге шаблона.
- Comparisons (int): Текущее количество сравнений символов, выполненных алгоритмом.
- PrefixFunction ([]int): Массив, содержащий текущие значения префикс-функции или длины совпадений для каждой позиции текста.

- HighlightPrefixIndex (int): Индекс символа в тексте, который должен быть подсвечен на фронтенде для визуализации текущего шага.

Эта структура позволяет фиксировать состояние алгоритма на каждом этапе, включая информацию о совпадениях, сдвигах и значениях префикс-функции, что делает возможной пошаговую анимацию процесса поиска.

Структура KMPResult агрегирует результаты выполнения алгоритма KMP и содержит данные, которые возвращаются фронтенду в ответ на запрос. Поля структуры включают:

- FailureFunction ([]int): Массив значений префикс-функции, построенный для шаблона.
- Steps ([]Step): Список всех шагов выполнения алгоритма, содержащих детализированную информацию для визуализации.
- Found (bool): Флаг, указывающий, были ли найдены вхождения шаблона в тексте.
- Positions ([]int): Список индексов в тексте, где были найдены вхождения шаблона.
- Comparisons (int): Общее количество сравнений символов, выполненных алгоритмом.
- Error (string, опционально): Сообщение об ошибке, если входные данные некорректны (например, пустой текст или шаблон).

Эти структуры обеспечивают структурированное представление данных, необходимых для обработки запросов и визуализации алгоритма на фронтенде.

3.2.2. Реализация алгоритма KMP

Алгоритм KMP реализован в двух основных функциях: buildFailureFunction и searchKMP. Эти функции обеспечивают эффективный поиск подстроки в тексте и генерацию данных для визуализации. Дополнительно реализована функция handleKMP для обработки HTTP-запросов.

Функция buildFailureFunction(pattern string) []int строит префикс-функцию (или функцию неудач) для заданного шаблона. Префикс-функция используется в алгоритме KMP для определения, на сколько позиций можно сдвинуть шаблон при несовпадении символов, избегая лишних сравнений.

Функция searchKMP(text string, pattern string) KMPResult выполняет поиск всех вхождений шаблона в тексте, используя алгоритм KMP, и генерирует пошаговые данные для визуализации. Она возвращает структуру KMPResult, содержащую результаты поиска и информацию о каждом шаге.

Функция handleKMP(w http.ResponseWriter, r *http.Request) обрабатывает HTTP POST-запросы от фронтенда, выполняет алгоритм KMP и возвращает результаты в формате JSON.

3.3. Реализация фронтенда

Фронтенд приложения реализован с использованием HTML, CSS и JavaScript и представляет собой интерактивный веб-интерфейс для визуализации работы алгоритма Кнута-Морриса-Пратта (KMP). Основная задача фронтенда — предоставить пользователю удобный способ ввода данных (текста и шаблона), управления анимацией и наблюдения за пошаговым выполнением алгоритма. Визуализация включает отображение текста, шаблона, текущего состояния сравнений, префикс-функции и результатов поиска. Данный раздел подробно описывает структуру HTML-страницы, стилизацию, логику JavaScript, а также особенности реализации.

3.3.1. Структура HTML-страницы

HTML-страница является основой фронтенда и содержит элементы для ввода данных, управления анимацией и отображения результатов. Основные компоненты страницы включают:

- Заголовок страницы: Установлен как "KMP Algorithm Visualization", что отражает назначение приложения — визуализация алгоритма KMP.
- Блок управления (div.controls):

    - Поля ввода (input[type="text"]) для текста и шаблона с начальными значениями (aababcabcdabcdeabcdef и abcdef соответственно).
    - Ползунок (input[type="range"]) для настройки скорости анимации (от 50 до 3000 мс с шагом 100 мс, начальное значение — 500 мс).
    - Кнопки управления анимацией: "Start Animation", "Pause", "Reset".
    - Кнопки пошагового управления: "Previous Step" и "Next Step", а также индикатор текущего шага (span#step-counter).
- Область канваса (div.canvas-container):
    - Элемент <canvas> с идентификатором canvas для отрисовки текста, шаблона и текущего состояния алгоритма.
    - Контейнер с горизонтальной прокруткой для поддержки длинных строк.
- Панель статуса (div#status):
    - Отображает текстовое описание текущего шага, включая индексы, результат сравнения и общее количество сравнений.
- Контейнер префикс-функции (div#prefix-container):
    - Содержит заголовок ("Prefix Function") и динамически генерируемый список значений префикс-функции для каждой позиции текста.
- Контейнер результатов (div#answer-container):
    - Отображает индексы вхождений шаблона в текст, полученные от бэкенда.

Структура HTML организована с использованием семантических элементов и классов для упрощения стилизации и взаимодействия через JavaScript. Все элементы управления сгруппированы в блок .controls, что обеспечивает логичную компоновку интерфейса.

3.3.2. Логика работы JavaScript

Логика фронтенда реализована на JavaScript и отвечает за взаимодействие с пользователем, отправку запросов к бэкенду, обработку данных и отрисовку визуализации. Основные функции JavaScript описаны ниже.

Функция resizeCanvas() Динамически изменяет размеры элемента <canvas> в зависимости от длины текста и шаблона, а также ширины контейнера.

Функция drawCharBox(x, y, value, isMismatch, isCurrent) отрисовывает одну ячейку с символом текста или шаблона.

Функция drawTextAndPattern(stepIndex) отрисовывает текущее состояние текста, шаблона, префикс-функции и статуса для заданного шага алгоритма.

ЗАКЛЮЧЕНИЕ

Разработанное приложение эффективно визуализирует алгоритм Кнута-Морриса-Пратта (KMP) для поиска подстроки. Бэкенд на Go обеспечивает высокую производительность O(n+m), надежную обработку запросов и генерацию пошаговых данных через структуры Step и KMPResult. Фронтенд, реализованный на HTML, CSS и JavaScript, предоставляет интерактивный интерфейс с динамической отрисовкой, управлением анимацией и наглядным отображением процесса. Приложение успешно решает образовательные задачи, помогая понять алгоритм KMP. Возможные улучшения включают поддержку кодировок, оптимизацию для длинных строк и адаптацию для мобильных устройств. Проект сочетает функциональность, производительность и удобство, соответствуя требованиям курсовой работы и имея потенциал для дальнейшего развития.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Программа Построение и анализ алгоритмов / URL: https://se.moevm.info/doku.php/courses:algorithms_building_and_analysis:lectures
2. Кнут-Мориссон-Прат / URL: https://ru.wikipedia.org/wiki/Алгоритм_Кнута_—_Морриса_Пратта

ПРИЛОЖЕНИЕ А

НАЗВАНИЕ ФАЙЛА: MAIN.GO

```
package main

import (
    "encoding/json"
    "fmt"
    "html/template"
    "log"
    "net/http"
)

// Step представляет один шаг выполнения алгоритма КМР
type Step struct {
    TextIndex            int    `json:"textIndex"`
    PatternIndex         int    `json:"patternIndex"`
    Match                bool   `json:"match"`
    Shift                bool   `json:"shift"`
    Status               string `json:"status"`
    FailureValue         int    `json:"failureValue,omitempty"`
    Comparisons          int    `json:"comparisons"`
    PrefixFunction       []int  `json:"prefixFunction,omitempty"`
// Динамическая информация о совпадениях
    HighlightPrefixIndex int
 `json:"highlightPrefixIndex,omitempty"`
}

// KMPResult представляет результат работы алгоритма КМР с шагами
type KMPResult struct {
    FailureFunction []int  `json:"failureFunction"`
    Steps           []Step `json:"steps"`
    Found           bool   `json:"found"`
    Positions       []int  `json:"positions"`
    Comparisons     int    `json:"comparisons"`
    Error           string `json:"error,omitempty"`
```

```
}

// buildFailureFunction строит функцию префиксов для подстроки
func buildFailureFunction(pattern string) []int {
    m := len(pattern)
    failure := make([]int, m)
    failure[0] = 0
    j := 0

    for i := 1; i < m; i++ {
        if pattern[i] == pattern[j] {
            j++
            failure[i] = j
        } else {
            for j > 0 && pattern[i] != pattern[j] {
                j = failure[j-1]
            }
            if pattern[i] == pattern[j] {
                j++
            }
            failure[i] = j
        }
    }
    return failure
}

// searchKMP выполняет поиск подстроки в тексте с шагами для
анимации
func searchKMP(text, pattern string) KMPResult {
    if pattern == "" {
        return KMPResult{Error: "Pattern cannot be empty"}
    }
    if text == "" {
        return KMPResult{Error: "Text cannot be empty"}
    }
```

```
    failure := buildFailureFunction(pattern)
    positions := []int{}
    steps := []Step{}
    comparisons := 0
    i := 0 // индекс в тексте
    j := 0 // индекс в подстроке
    n := len(text)
    m := len(pattern)

    // Создаем массив для хранения максимальных длин совпадений
для каждого символа текста
    maxPrefix := make([]int, n)

    for i < n {
        comparisons++
        match := text[i] == pattern[j]
        step := Step{
            TextIndex:      i,
            PatternIndex:   j,
            Match:          match,
            Shift:          false,
            Comparisons:    comparisons,
            PrefixFunction: make([]int, n),
        }

        // Копируем предыдущие максимальные значения
        copy(step.PrefixFunction, maxPrefix)

        // Устанавливаем HighlightPrefixIndex для текущей позиции
текста
        step.HighlightPrefixIndex = i // Подсвечиваем текущий
индекс текста

        if match {
```

```
            // Обновляем максимальную длину совпадения для текущей
позиции
            if j+1 > maxPrefix[i] {
                maxPrefix[i] = j + 1
            }
            step.PrefixFunction[i] = maxPrefix[i]
            step.Status = fmt.Sprintf("Comparing text[%d]='%c'
with pattern[%d]='%c': match found. Advancing to text[%d] and
pattern[%d].",
                i, text[i], j, pattern[j], i+1, j+1)
            steps = append(steps, step)
            i++
            j++
            if j == m {
                positions = append(positions, i-j)
                // Обновляем все позиции, где было полное
совпадение
                for k := i - m; k < i; k++ {
                    if k >= 0 && k < n {
                        maxPrefix[k] = k - (i - m) + 1 // Длина
совпадения от начала
                    }
                }
                // Шаг для полного совпадения
                step = Step{
                    TextIndex:    i - 1,
                    PatternIndex: j - 1,
                    Match:        true,
                    Shift:        true,
                    Status: fmt.Sprintf("Full pattern match found
at text index %d! Using failure function value %d to shift pattern
to pattern[%d].",
                        i-j, failure[j-1], failure[j-1]),
                    FailureValue:         failure[j-1],
                    Comparisons:          comparisons,
```

```
                    PrefixFunction:       make([]int, n),
                    HighlightPrefixIndex: -1, // Убираем подсветку
на последнем шаге
                }
                copy(step.PrefixFunction, maxPrefix)
                steps = append(steps, step)
                j = failure[j-1]
            }
        } else if j > 0 {
            step.Status = fmt.Sprintf("Mismatch at text[%d]='%c'
and pattern[%d]='%c'. Using failure function value %d to shift
pattern to pattern[%d].",
                i, text[i], j, pattern[j], failure[j-1],
failure[j-1])
            step.FailureValue = failure[j-1]
            step.HighlightPrefixIndex = i // Подсвечиваем текущий
индекс при несоответствии
            steps = append(steps, step)
            j = failure[j-1]
            step = Step{
                TextIndex:    i,
                PatternIndex: j,
                Match:        false,
                Shift:        true,
                Status: fmt.Sprintf("Pattern shifted to align at
pattern[%d] with text[%d]='%c' based on failure function.",
                    j, i, text[i]),
                Comparisons:          comparisons,
                PrefixFunction:       make([]int, n),
                HighlightPrefixIndex: i, // Подсвечиваем текущий
индекс
            }
            copy(step.PrefixFunction, maxPrefix)
            steps = append(steps, step)
        } else {
```

```
            step.Status = fmt.Sprintf("Mismatch at text[%d]='%c'
and pattern[0]='%c'. No prefix to use, advancing to text[%d].",
                i, text[i], pattern[0], i+1)
            step.HighlightPrefixIndex = i // Подсвечиваем текущий
индекс
            steps = append(steps, step)
            i++
        }
    }

    return KMPResult{
        FailureFunction: failure,
        Steps:           steps,
        Found:           len(positions) > 0,
        Positions:       positions,
        Comparisons:     comparisons,
    }
}

// handleIndex обслуживает главную страницу
func handleIndex(w http.ResponseWriter, r *http.Request) {
    tmpl, err := template.ParseFiles("index.html")
    if err != nil {
        http.Error(w, "Internal Server Error",
http.StatusInternalServerError)
        log.Println(err)
        return
    }
    tmpl.Execute(w, nil)
}

// handleKMP обрабатывает запросы к API для выполнения KMP
func handleKMP(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
```

```
        http.Error(w, "Method not allowed",
http.StatusMethodNotAllowed)
        return
    }

    var input struct {
        Text    string `json:"text"`
        Pattern string `json:"pattern"`
    }
    err := json.NewDecoder(r.Body).Decode(&input)
    if err != nil {
        http.Error(w, "Bad request", http.StatusBadRequest)
        return
    }

    result := searchKMP(input.Text, input.Pattern)
    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(result)
}

func main() {
    http.HandleFunc("/", handleIndex)
    http.HandleFunc("/kmp", handleKMP)

    fmt.Println("Server starting on :8080")
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

НАЗВАНИЕ ФАЙЛА: INDEX.HTML

```
<!DOCTYPE HTML>
<HTML LANG="EN">
<HEAD>
    <META CHARSET="UTF-8">
    <TITLE>KMP ALGORITHM VISUALIZATION</TITLE>
    <STYLE>
```

```
        BODY {
            FONT-FAMILY: 'SEGOE UI', TAHOMA, GENEVA, VERDANA,
SANS-SERIF;
            MARGIN: 30PX;
            BACKGROUND-COLOR: #F4F4F4;
            COLOR: #333;
        }
        .CONTROLS {
            BACKGROUND-COLOR: #FFF;
            PADDING: 20PX;
            BORDER-RADIUS: 8PX;
            BOX-SHADOW: 0 2PX 4PX RGBA(0, 0, 0, 0.1);
            MARGIN-BOTTOM: 20PX;
            DISPLAY: FLEX;
            GAP: 20PX;
            ALIGN-ITEMS: CENTER;
            FLEX-WRAP: WRAP;
            JUSTIFY-CONTENT: SPACE-BETWEEN;
        }
        .CONTROLS > DIV {
            DISPLAY: FLEX;
            ALIGN-ITEMS: CENTER;
        }
        .CONTROLS LABEL {
            FONT-WEIGHT: BOLD;
            COLOR: #555;
            MARGIN-RIGHT: 10PX;
        }
        .CONTROLS INPUT[TYPE="TEXT"],
        .CONTROLS INPUT[TYPE="RANGE"] {
            PADDING: 10PX;
            BORDER: 1PX SOLID #DDD;
            BORDER-RADIUS: 4PX;
            FONT-SIZE: 16PX;
            MIN-WIDTH: 150PX;
```

```
}
.CONTROLS INPUT[TYPE="RANGE"] {
    WIDTH: 200PX;
}
.CONTROLS-BUTTONS {
    DISPLAY: FLEX;
    GAP: 10PX;
}
.CONTROLS-BUTTONS BUTTON {
    BACKGROUND-COLOR: #5CB85C;
    COLOR: WHITE;
    BORDER: NONE;
    PADDING: 10PX 15PX;
    BORDER-RADIUS: 4PX;
    CURSOR: POINTER;
    FONT-SIZE: 16PX;
    TRANSITION: BACKGROUND-COLOR 0.3S EASE;
}
.CONTROLS-BUTTONS BUTTON:HOVER {
    BACKGROUND-COLOR: #4CAE4C;
}
.STEP-CONTROLS {
    DISPLAY: FLEX;
    GAP: 10PX;
    ALIGN-ITEMS: CENTER;
}
.STEP-CONTROLS BUTTON {
    BACKGROUND-COLOR: #007BFF;
    COLOR: WHITE;
    BORDER: NONE;
    PADDING: 8PX 12PX;
    BORDER-RADIUS: 4PX;
    CURSOR: POINTER;
    FONT-SIZE: 14PX;
    TRANSITION: BACKGROUND-COLOR 0.3S EASE;
```

```
}
.STEP-CONTROLS BUTTON:HOVER {
    BACKGROUND-COLOR: #0056B3;
}
.CANVAS-CONTAINER {
    WIDTH: 100%;
    OVERFLOW-X: AUTO;
    MARGIN-BOTTOM: 20PX;
    BORDER: 1PX SOLID #CCC;
    BACKGROUND-COLOR: #FFF;
    BOX-SHADOW: 0 2PX 4PX RGBA(0, 0, 0, 0.1);
}
CANVAS {
    DISPLAY: BLOCK;
    BACKGROUND-COLOR: #FFF;
}
#STATUS {
    BACKGROUND-COLOR: #E9ECEF;
    PADDING: 15PX;
    BORDER-RADIUS: 4PX;
    MARGIN-TOP: 10PX;
    FONT-SIZE: 16PX;
    COLOR: #495057;
    WHITE-SPACE: PRE-WRAP;
}
#SPEED-LABEL {
    MARGIN-LEFT: 10PX;
    COLOR: #555;
}
#PREFIX-CONTAINER {
    BACKGROUND-COLOR: #FFF;
    BORDER: 1PX SOLID #DDD;
    PADDING: 15PX;
    BORDER-RADIUS: 4PX;
    MARGIN-TOP: 20PX;
```

```
    BOX-SHADOW: 0 2PX 4PX RGBA(0, 0, 0, 0.1);
    MAX-WIDTH: 100%;
    OVERFLOW-X: AUTO;
}
#PREFIX-LABEL {
    FONT-WEIGHT: BOLD;
    COLOR: #555;
    MARGIN-BOTTOM: 15PX;
    DISPLAY: BLOCK;
}
#PREFIX-FUNCTION-CONTAINER {
    DISPLAY: FLEX;
    ALIGN-ITEMS: CENTER;
    GAP: 8PX;
    OVERFLOW-X: AUTO;
    WIDTH: 100%;
    PADDING-BOTTOM: 5PX;
}
.PREFIX-BOX {
    WIDTH: 35PX;
    HEIGHT: 35PX;
    BORDER: 1PX SOLID #CCC;
    DISPLAY: FLEX;
    JUSTIFY-CONTENT: CENTER;
    ALIGN-ITEMS: CENTER;
    FONT-FAMILY: 'COURIER NEW', COURIER, MONOSPACE;
    FONT-SIZE: 16PX;
    BACKGROUND-COLOR: #F9F9F9;
    BORDER-RADIUS: 4PX;
    POSITION: RELATIVE;
    FLEX-SHRINK: 0;
}
.PREFIX-INDEX {
    POSITION: ABSOLUTE;
    TOP: -20PX;
```

```
    FONT-SIZE: 14PX;
    COLOR: #777;
}
.PREFIX-BOTTOM-INDEX {
    POSITION: ABSOLUTE;
    BOTTOM: -20PX;
    FONT-SIZE: 12PX;
    COLOR: #777;
    WIDTH: 100%;
    TEXT-ALIGN: CENTER;
}
#ANSWER-CONTAINER {
    BACKGROUND-COLOR: #FFF;
    BORDER: 1PX SOLID #DDD;
    PADDING: 15PX;
    BORDER-RADIUS: 4PX;
    MARGIN-TOP: 20PX;
    BOX-SHADOW: 0 2PX 4PX RGBA(0, 0, 0, 0.1);
}
#ANSWER-LABEL {
    FONT-WEIGHT: BOLD;
    COLOR: #555;
    MARGIN-BOTTOM: 10PX;
    DISPLAY: BLOCK;
}
#ANSWER {
    FONT-FAMILY: 'COURIER NEW', COURIER, MONOSPACE;
    FONT-SIZE: 16PX;
    COLOR: #383D41;
    BACKGROUND-COLOR: #E0F7FA;
    PADDING: 8PX;
    BORDER-RADIUS: 4PX;
}
.TEXT-INDEX {
    POSITION: ABSOLUTE;
```

```
            BOTTOM: -25PX;
            FONT-SIZE: 14PX;
            COLOR: #777;
            WIDTH: 35PX;
            TEXT-ALIGN: CENTER;
        }
    </STYLE>
</HEAD>
<BODY>
    <H1>KMP ALGORITHM VISUALIZATION</H1>
    <DIV CLASS="CONTROLS">
        <DIV>
            <LABEL FOR="TEXT">TEXT:</LABEL>
            <INPUT TYPE="TEXT" ID="TEXT"
VALUE="AABABCABCDABCDEABCDEF">
        </DIV>
        <DIV>
            <LABEL FOR="PATTERN">PATTERN:</LABEL>
            <INPUT TYPE="TEXT" ID="PATTERN" VALUE="ABCDEF">
        </DIV>
        <DIV>
            <LABEL FOR="SPEED">ANIMATION SPEED:</LABEL>
            <INPUT TYPE="RANGE" ID="SPEED" MIN="50" MAX="3000"
VALUE="500" STEP="100">
            <SPAN ID="SPEED-LABEL">500 MS</SPAN>
        </DIV>
        <DIV CLASS="CONTROLS-BUTTONS">
            <BUTTON ONCLICK="STARTANIMATION()">START
ANIMATION</BUTTON>
            <BUTTON ONCLICK="PAUSEANIMATION()">PAUSE</BUTTON>
            <BUTTON ONCLICK="RESET()">RESET</BUTTON>
        </DIV>
        <DIV CLASS="STEP-CONTROLS">
            <BUTTON ONCLICK="PREVSTEP()">PREVIOUS STEP</BUTTON>
            <SPAN ID="STEP-COUNTER">STEP: 0/0</SPAN>
```

```
        <BUTTON ONCLICK="NEXTSTEP()">NEXT STEP</BUTTON>
    </DIV>
</DIV>
<DIV CLASS="CANVAS-CONTAINER">
    <CANVAS ID="CANVAS" WIDTH="800" HEIGHT="220"></CANVAS>
</DIV>
<DIV ID="STATUS"></DIV>

<DIV ID="PREFIX-CONTAINER">
    <DIV ID="PREFIX-LABEL">PREFIX FUNCTION:</DIV>
    <DIV ID="PREFIX-FUNCTION-CONTAINER"></DIV>
</DIV>

<DIV ID="ANSWER-CONTAINER">
    <DIV ID="ANSWER-LABEL">FOUND AT INDICES:</DIV>
    <DIV ID="ANSWER"></DIV>
</DIV>

<SCRIPT>
    CONST CANVAS = DOCUMENT.GETELEMENTBYID("CANVAS");
    CONST CTX = CANVAS.GETCONTEXT("2D");
    CONST BOX_WIDTH = 35;
    CONST BOX_HEIGHT = 35;
    CONST MARGIN_X = 40;
    CONST MARGIN_Y = 40;
    CONST ROW_SPACING = 25;
    LET STEPS = [];
    LET CURRENTSTEP = 0;
    LET TEXT = "";
    LET PATTERN = "";
    LET ANIMATIONID = NULL;
    LET ISANIMATING = FALSE;

    // УСТАНОВКА ДИНАМИЧЕСКОГО РАЗМЕРА CANVAS
    FUNCTION RESIZECANVAS() {
```

```
            CONST CONTAINER = DOCUMENT.QUERYSELECTOR('.CANVAS-
CONTAINER');
            CONST TEXTINPUT =
DOCUMENT.GETELEMENTBYID("TEXT").VALUE;
            CONST PATTERNINPUT =
DOCUMENT.GETELEMENTBYID("PATTERN").VALUE;

            // РАССЧИТЫВАЕМ НЕОБХОДИМУЮ ШИРИНУ CANVAS
            CONST MAXLENGTH = MATH.MAX(TEXTINPUT.LENGTH,
PATTERNINPUT.LENGTH);
            CONST REQUIREDWIDTH = MARGIN_X * 2 + MAXLENGTH *
BOX_WIDTH;

            // УСТАНАВЛИВАЕМ РАЗМЕРЫ CANVAS
            CANVAS.WIDTH = MATH.MAX(REQUIREDWIDTH,
CONTAINER.CLIENTWIDTH);
            CANVAS.HEIGHT = 220;

            // МАСШТАБИРОВАНИЕ ДЛЯ HIDPI ДИСПЛЕЕВ
            CONST DPR = WINDOW.DEVICEPIXELRATIO || 1;
            CONST RECT = CANVAS.GETBOUNDINGCLIENTRECT();
            CANVAS.WIDTH = RECT.WIDTH * DPR;
            CANVAS.HEIGHT = RECT.HEIGHT * DPR;
            CANVAS.STYLE.WIDTH = `${RECT.WIDTH}PX`;
            CANVAS.STYLE.HEIGHT = `${RECT.HEIGHT}PX`;
            CTX.SCALE(DPR, DPR);
        }

        // ИНИЦИАЛИЗАЦИЯ РАЗМЕРА CANVAS
        RESIZECANVAS();
        WINDOW.ADDEVENTLISTENER('RESIZE', RESIZECANVAS);

        // ОБНОВЛЕНИЕ МЕТКИ СКОРОСТИ
        CONST SPEEDINPUT = DOCUMENT.GETELEMENTBYID("SPEED");
        CONST SPEEDLABEL = DOCUMENT.GETELEMENTBYID("SPEED-LABEL");
```

```
        SPEEDINPUT.ADDEVENTLISTENER("INPUT", () => {
            SPEEDLABEL.TEXTCONTENT = `${SPEEDINPUT.VALUE} MS`;
        });

        FUNCTION UPDATESTEPCOUNTER() {
            DOCUMENT.GETELEMENTBYID("STEP-COUNTER").INNERHTML =
`STEP: ${CURRENTSTEP + 1} / ${STEPS.LENGTH}`;
        }

        FUNCTION DRAWCHARBOX(X, Y, VALUE, ISMISMATCH, ISCURRENT) {
            CONST BACKGROUNDCOLOR = ISMISMATCH ? "#FFDDDD" :
(ISCURRENT ? "#DCDCDC" : "#F9F9F9");
            CTX.FILLSTYLE = BACKGROUNDCOLOR;
            CTX.STROKESTYLE = "#CCC";
            CTX.LINEWIDTH = 1;
            CTX.FILLRECT(X, Y, BOX_WIDTH, BOX_HEIGHT);
            CTX.STROKERECT(X, Y, BOX_WIDTH, BOX_HEIGHT);

            // HIGHLIGHT MISMATCH WITH A STRONGER RED BORDER
            IF (ISMISMATCH) {
                CTX.LINEWIDTH = 2;
                CTX.STROKESTYLE = "#FF4D4D";
                CTX.STROKERECT(X - 1, Y - 1, BOX_WIDTH + 2,
BOX_HEIGHT + 2);
                CTX.LINEWIDTH = 1;
                CTX.STROKESTYLE = "#CCC";
            }

            CTX.FILLSTYLE = "#333";
            CTX.FONT = "18PX 'COURIER NEW', COURIER, MONOSPACE";
            CTX.TEXTALIGN = "CENTER";
            CTX.FILLTEXT(VALUE, X + BOX_WIDTH / 2, Y + BOX_HEIGHT
/ 2 + 6);
            CTX.TEXTALIGN = "LEFT";
        }
```

```
        FUNCTION DRAWTEXTANDPATTERN(STEPINDEX) {
            CTX.CLEARRECT(0, 0, CANVAS.WIDTH, CANVAS.HEIGHT);
            CTX.FONT = "16PX MONOSPACE";
            CTX.TEXTALIGN = "LEFT";

            IF (STEPS.LENGTH === 0 || STEPINDEX < 0 || STEPINDEX
>= STEPS.LENGTH) RETURN;

            CONST STEP = STEPS[STEPINDEX];
            CONST TEXTINDEX = STEP.TEXTINDEX;
            CONST PATTERNINDEX = STEP.PATTERNINDEX;
            CONST SHIFT = TEXTINDEX - PATTERNINDEX;

            // ОТРИСОВКА ТЕКСТА
            FOR (LET I = 0; I < TEXT.LENGTH; I++) {
                CONST X = MARGIN_X + I * BOX_WIDTH;
                CONST Y = MARGIN_Y;
                CONST ISCURRENT = I === TEXTINDEX;
                CONST ISMISMATCH = ISCURRENT && PATTERNINDEX >= 0
&&
                                    TEXT[TEXTINDEX] !==
PATTERN[PATTERNINDEX];
                DRAWCHARBOX(X, Y, TEXT[I], ISMISMATCH);
            }

            // ОТРИСОВКА ПОДСТРОКИ С УЧЕТОМ СДВИГА
            FOR (LET I = 0; I < PATTERN.LENGTH; I++) {
                CONST X = MARGIN_X + (I + SHIFT) * BOX_WIDTH;
                CONST Y = MARGIN_Y + BOX_HEIGHT + ROW_SPACING +
30;
                CONST ISCURRENT = I === PATTERNINDEX;
                CONST ISMISMATCH = ISCURRENT && PATTERNINDEX >= 0
&&
```

```
                                TEXT[TEXTINDEX] !==
PATTERN[PATTERNINDEX];
                DRAWCHARBOX(X, Y, PATTERN[I], ISMISMATCH);
            }

            // ОТРИСОВКА ГОЛУБЫХ РАМОК ДЛЯ ТЕКУЩИХ СИМВОЛОВ ПОСЛЕ
ВСЕХ КВАДРАТОВ
            FOR (LET I = 0; I < TEXT.LENGTH; I++) {
                CONST X = MARGIN_X + I * BOX_WIDTH;
                CONST Y = MARGIN_Y;
                CONST ISCURRENT = I === TEXTINDEX;
                IF (ISCURRENT) {
                    CTX.STROKESTYLE = "#007BFF";
                    CTX.LINEWIDTH = 3;
                    CTX.STROKERECT(X - 1.5, Y - 1.5, BOX_WIDTH +
3, BOX_HEIGHT + 3);
                }
            }

            FOR (LET I = 0; I < PATTERN.LENGTH; I++) {
                CONST X = MARGIN_X + (I + SHIFT) * BOX_WIDTH;
                CONST Y = MARGIN_Y + BOX_HEIGHT + ROW_SPACING +
30;
                CONST ISCURRENT = I === PATTERNINDEX;
                IF (ISCURRENT) {
                    CTX.STROKESTYLE = "#007BFF";
                    CTX.LINEWIDTH = 3;
                    CTX.STROKERECT(X - 1.5, Y - 1.5, BOX_WIDTH +
3, BOX_HEIGHT + 3);
                }
            }

            // ОТРИСОВКА ИНДЕКСОВ ПОД ТЕКСТОМ
            FOR (LET I = 0; I < TEXT.LENGTH; I++) {
                CONST X = MARGIN_X + I * BOX_WIDTH;
```

```
                CTX.FILLSTYLE = "#777";
                CTX.FONT = "14PX 'SEGOE UI', TAHOMA, GENEVA,
VERDANA, SANS-SERIF";
                CTX.TEXTALIGN = "CENTER";
                CTX.FILLTEXT(I, X + BOX_WIDTH / 2, MARGIN_Y +
BOX_HEIGHT + 20);
                CTX.TEXTALIGN = "LEFT";
            }

            // ОБНОВЛЕНИЕ СТАТУСА
            DOCUMENT.GETELEMENTBYID("STATUS").TEXTCONTENT =
                `STEP ${STEPINDEX + 1}: ${STEP.STATUS}
(COMPARISONS: ${STEP.COMPARISONS})`;

            // ОБНОВЛЕНИЕ МАССИВА ПРЕФИКС-ФУНКЦИИ
            CONST PREFIXCONTAINER =
DOCUMENT.GETELEMENTBYID("PREFIX-FUNCTION-CONTAINER");
            PREFIXCONTAINER.INNERHTML = "";
            IF (STEP.PREFIXFUNCTION) {
                STEP.PREFIXFUNCTION.FOREACH((VALUE, INDEX) => {
                    CONST WRAPPER = DOCUMENT.CREATEELEMENT("DIV");
                    WRAPPER.STYLE.DISPLAY = "FLEX";
                    WRAPPER.STYLE.FLEXDIRECTION = "COLUMN";
                    WRAPPER.STYLE.ALIGNITEMS = "CENTER";
                    WRAPPER.STYLE.MARGINRIGHT = "5PX";

                    CONST BOX = DOCUMENT.CREATEELEMENT("DIV");
                    BOX.CLASSLIST.ADD("PREFIX-BOX");
                    BOX.TEXTCONTENT = VALUE;

                    IF (STEP.HIGHLIGHTPREFIXINDEX !== UNDEFINED &&
STEP.HIGHLIGHTPREFIXINDEX === INDEX) {
                        BOX.STYLE.BACKGROUNDCOLOR = "#CCCCCC";
                        BOX.STYLE.BORDER = "2PX SOLID #007BFF";
                    } ELSE {
```

```
                    BOX.STYLE.BACKGROUNDCOLOR = "#F9F9F9";
                    BOX.STYLE.BORDER = "1PX SOLID #CCC";
                }

                CONST INDEXLABEL =
DOCUMENT.CREATEELEMENT("DIV");
                INDEXLABEL.CLASSLIST.ADD("PREFIX-INDEX");
                INDEXLABEL.TEXTCONTENT = INDEX;
                BOX.APPENDCHILD(INDEXLABEL);

                CONST BOTTOMINDEXLABEL =
DOCUMENT.CREATEELEMENT("DIV");
                BOTTOMINDEXLABEL.STYLE.FONTSIZE = "12PX";
                BOTTOMINDEXLABEL.STYLE.COLOR = "#777";
                BOTTOMINDEXLABEL.STYLE.MARGINTOP = "2PX";
                BOTTOMINDEXLABEL.TEXTCONTENT = INDEX;

                WRAPPER.APPENDCHILD(BOX);
                WRAPPER.APPENDCHILD(BOTTOMINDEXLABEL);
                PREFIXCONTAINER.APPENDCHILD(WRAPPER);
            });
        }

        UPDATESTEPCOUNTER();
    }

    FUNCTION ANIMATE() {
        IF (CURRENTSTEP >= STEPS.LENGTH) {
            PAUSEANIMATION();
            DOCUMENT.GETELEMENTBYID("STATUS").TEXTCONTENT +=
"\NANIMATION FINISHED.";
            RETURN;
        }

        DRAWTEXTANDPATTERN(CURRENTSTEP);
```

```
            IF (ISANIMATING) {
                CURRENTSTEP++;
                CONST DELAY = PARSEINT(SPEEDINPUT.VALUE);
                SETTIMEOUT(() => {
                    IF (CURRENTSTEP < STEPS.LENGTH) {
                        ANIMATIONID =
REQUESTANIMATIONFRAME(ANIMATE);
                    }
                }, DELAY);
            }
        }

        FUNCTION STARTANIMATION() {
            // ЕСЛИ ШАГИ ЕЩЕ НЕ ЗАГРУЖЕНЫ ИЛИ АНИМАЦИЯ ЗАВЕРШЕНА
            IF (STEPS.LENGTH === 0 || CURRENTSTEP >= STEPS.LENGTH)
{
                CONST TEXTINPUT =
DOCUMENT.GETELEMENTBYID("TEXT").VALUE;
                CONST PATTERNINPUT =
DOCUMENT.GETELEMENTBYID("PATTERN").VALUE;

                // ОБНОВЛЯЕМ РАЗМЕР CANVAS ПЕРЕД НАЧАЛОМ АНИМАЦИИ
                RESIZECANVAS();

                FETCH('/KMP', {
                    METHOD: 'POST',
                    HEADERS: { 'CONTENT-TYPE': 'APPLICATION/JSON'
},
                    BODY: JSON.STRINGIFY({ TEXT: TEXTINPUT,
PATTERN: PATTERNINPUT })
                })
                .THEN(RESPONSE => RESPONSE.JSON())
                .THEN(DATA => {
                    IF (DATA.ERROR) {
```

```
DOCUMENT.GETELEMENTBYID("STATUS").TEXTCONTENT = `ERROR:
${DATA.ERROR}`;
                        RETURN;
                    }

                    TEXT = TEXTINPUT;
                    PATTERN = PATTERNINPUT;
                    STEPS = DATA.STEPS;
                    CURRENTSTEP = 0;
                    ISANIMATING = TRUE;
                    IF (ANIMATIONID)
CANCELANIMATIONFRAME(ANIMATIONID);
                    ANIMATE();
                    DOCUMENT.GETELEMENTBYID("ANSWER").TEXTCONTENT
= DATA.POSITIONS.JOIN(", ");
                })
                .CATCH(ERR => {
                    CONSOLE.ERROR(ERR);
                    DOCUMENT.GETELEMENTBYID("STATUS").TEXTCONTENT
= "AN ERROR OCCURRED.";
                });
            } ELSE {
                // ПРОДОЛЖАЕМ АНИМАЦИЮ С ТЕКУЩЕГО ШАГА
                ISANIMATING = TRUE;
                IF (ANIMATIONID)
CANCELANIMATIONFRAME(ANIMATIONID);
                ANIMATE();
            }
        }

        FUNCTION PAUSEANIMATION() {
            ISANIMATING = FALSE;
            IF (ANIMATIONID) {
                CANCELANIMATIONFRAME(ANIMATIONID);
```

```
            ANIMATIONID = NULL;
        }
    }

    FUNCTION PREVSTEP() {
        PAUSEANIMATION();
        IF (CURRENTSTEP > 0) {
            CURRENTSTEP--;
            DRAWTEXTANDPATTERN(CURRENTSTEP);
        }
    }

    FUNCTION NEXTSTEP() {
        PAUSEANIMATION();
        IF (CURRENTSTEP < STEPS.LENGTH - 1) {
            CURRENTSTEP++;
            DRAWTEXTANDPATTERN(CURRENTSTEP);
        }
    }

    FUNCTION RESET() {
        DOCUMENT.GETELEMENTBYID("TEXT").VALUE =
"AABABCABCDABCDEABCDEF";
        DOCUMENT.GETELEMENTBYID("PATTERN").VALUE = "ABCDEF";
        DOCUMENT.GETELEMENTBYID("SPEED").VALUE = "500";
        SPEEDLABEL.TEXTCONTENT = "500 MS";
        DOCUMENT.GETELEMENTBYID("STATUS").TEXTCONTENT = "";
        DOCUMENT.GETELEMENTBYID("ANSWER").TEXTCONTENT = "";
        DOCUMENT.GETELEMENTBYID("PREFIX-FUNCTION-
CONTAINER").INNERHTML = "";
        CTX.CLEARRECT(0, 0, CANVAS.WIDTH, CANVAS.HEIGHT);
        PAUSEANIMATION();
        STEPS = [];
        CURRENTSTEP = 0;
        UPDATESTEPCOUNTER();
```

```
            RESIZECANVAS();
        }

        // INITIAL STEP COUNTER UPDATE
        UPDATESTEPCOUNTER();
    </SCRIPT>
</BODY>
</HTML>
```